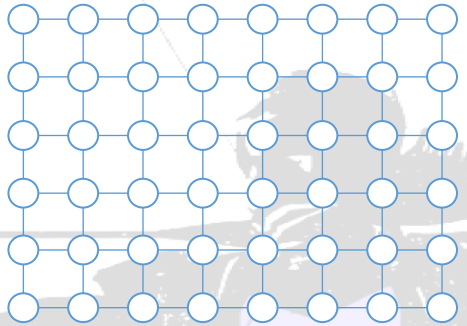
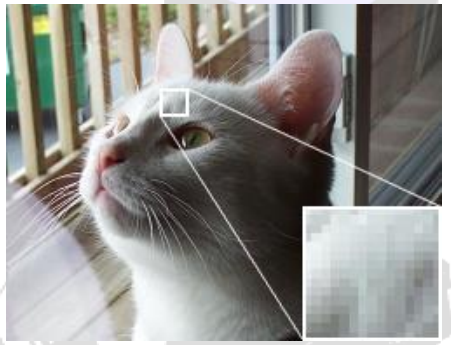
Line Algorithms

Instructor: Tony Diaz

What does “image” mean?

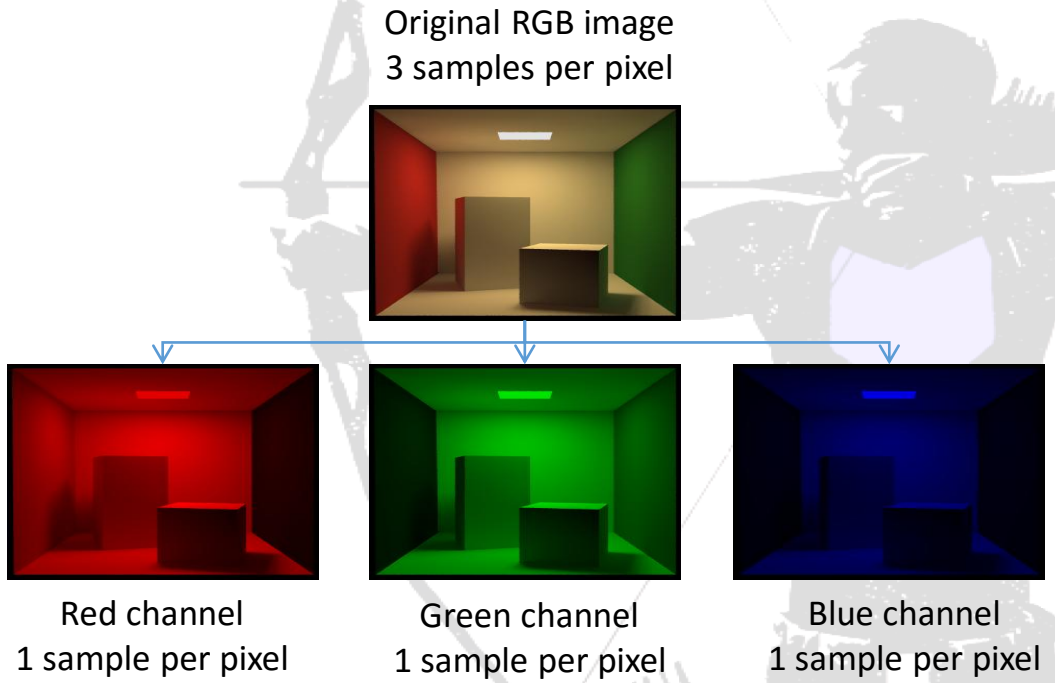
- A 2D domain with samples at regular points (almost always a rectilinear grid)
  - Can have multiple values sampled per point
  - Meaning of samples depend on the application (red, green, blue, opacity, depth, etc.)
- Units also depend on the application
  - e.g., a computed int or float to be mapped to voltage needed for display of a pixel on a screen
    - e.g., as a physical measurement of incoming light (e.g., a camera pixel sensor)





What is a “channel”?

- A channel is all the samples of a particular type
- The red channel of an RGB image is an image containing just the red samples
- TV channels are tuned to different frequencies of electromagnetic waves
  - In a similar sense, RGB channels are “tuned” to different frequencies in the visible spectrum



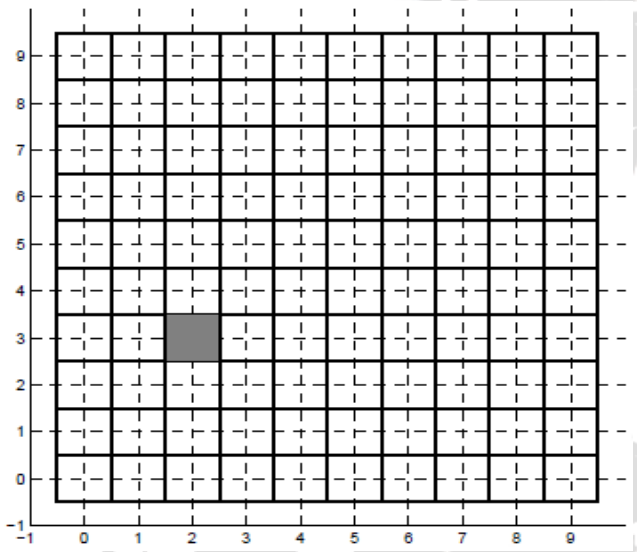
The “alpha” channel

- In addition to the R, G, and B channels of an image, add a fourth channel called α (transparency/opacity/translucency)
- Alpha varies between 0 and 1
  - Value of 1 represents a completely opaque pixel, one you cannot see through
  - Value of 0 is a completely transparent pixel
  - $0 < \alpha < 1$ => translucency
- Useful for blending images
  - Images with higher alpha values are less transparent
  - Linear interpolation ($\alpha X + (1- \alpha)Y$) or full Porter-Duff compositing algebra

The orange box is drawn on top of the purple box using $\alpha = 0.8$

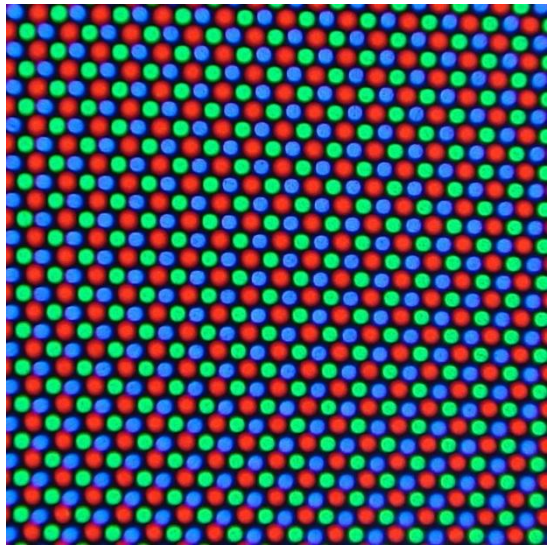
Modeling an Image

- Model a one-channel $n \times k$ image as the function $u(i,j)$ from pairs of integers to real numbers
  - i and j are integers such that $0 \le i < n$ and $0 \le j < k$
- Associate each pixel value $u(i,j)$ to small area around display location with coordinates (i,j)
- A pixel here looks like a square centered over the sample point, but it's just a scalar value and the actual geometry of its screen appearance varies by device
  - Roughly circular spot on CRT
  - Rectangular on LCD panel

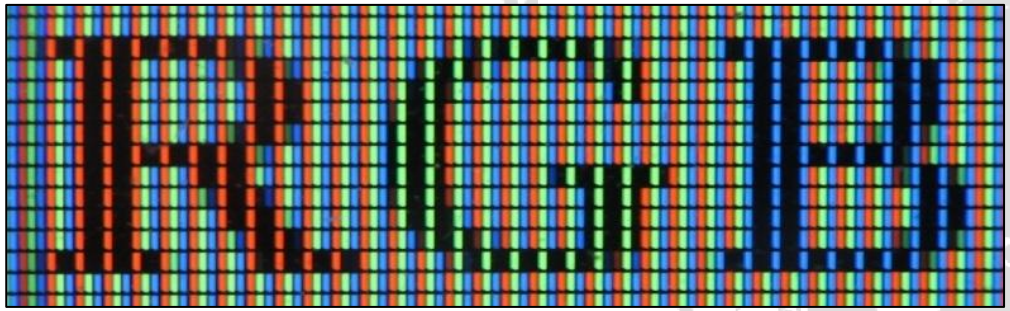


Pixels

- Pixels are point samples, not “squares” or “dots”
- Point samples reconstructed for display (often using multiple subpixels for primary colors)



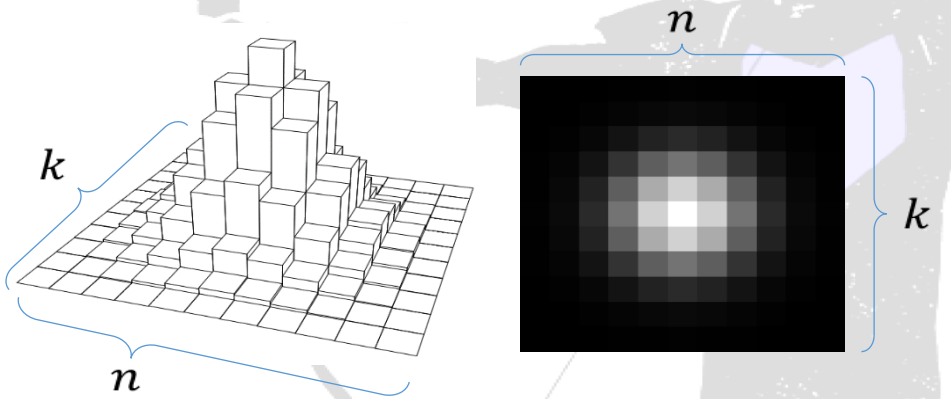
Close-up of a CRT screen



Close-up of an LCD screen

Discrete vs Continuous Images

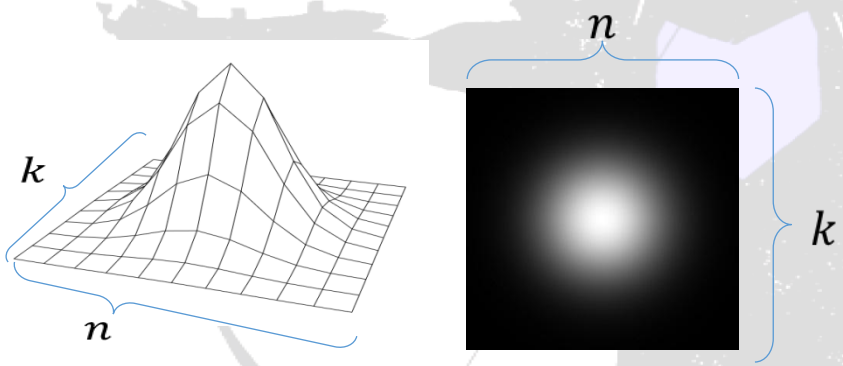
- Two kinds of images
  - Discrete
  - Continuous
- Discrete image
  - Function from $\mathbb{Z}^2$ to $\mathbb{R}$
  - How images are stored in memory
  - The kind of images we generally deal with as computer scientists



Discrete image $u(i, j)$

Discrete vs Continuous Images

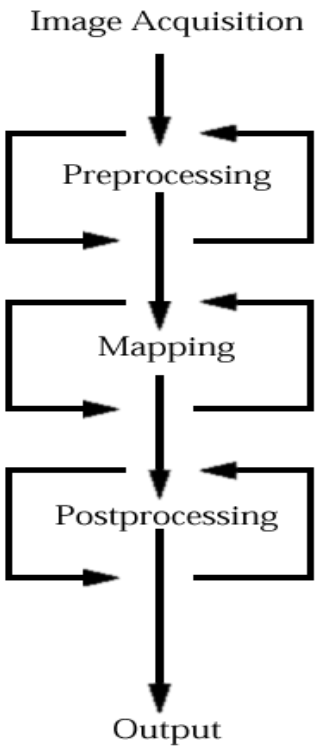
- Continuous image
  - Function from $\mathbb{R}^2$ to $\mathbb{R}$
  - Images in the real world
  - "Continuous" refers to the domain, not the values (discontinuities could still exist)
- Example: Gaussian distribution
  - i_0 and j_0 are the center of the Gaussian
  - $u:\mathbb{Z}^2\rightarrow\mathbb{R}$, $u(i,j)=e^{-(i-i_0)^2-(j-j_0)^2}$
  - $v:\mathbb{R}^2\rightarrow\mathbb{R}$, $v(i,j)=e^{-(i-i_0)^2-(j-j_0)^2}$
  - $i_0=(n-1)/2$ and $j_0=(k-1)/2$ (n odd)
  - Here n=11 and k=11



Continuous image $v(i,j)$

Idealized 5 Stage Pipeline of Image Processing

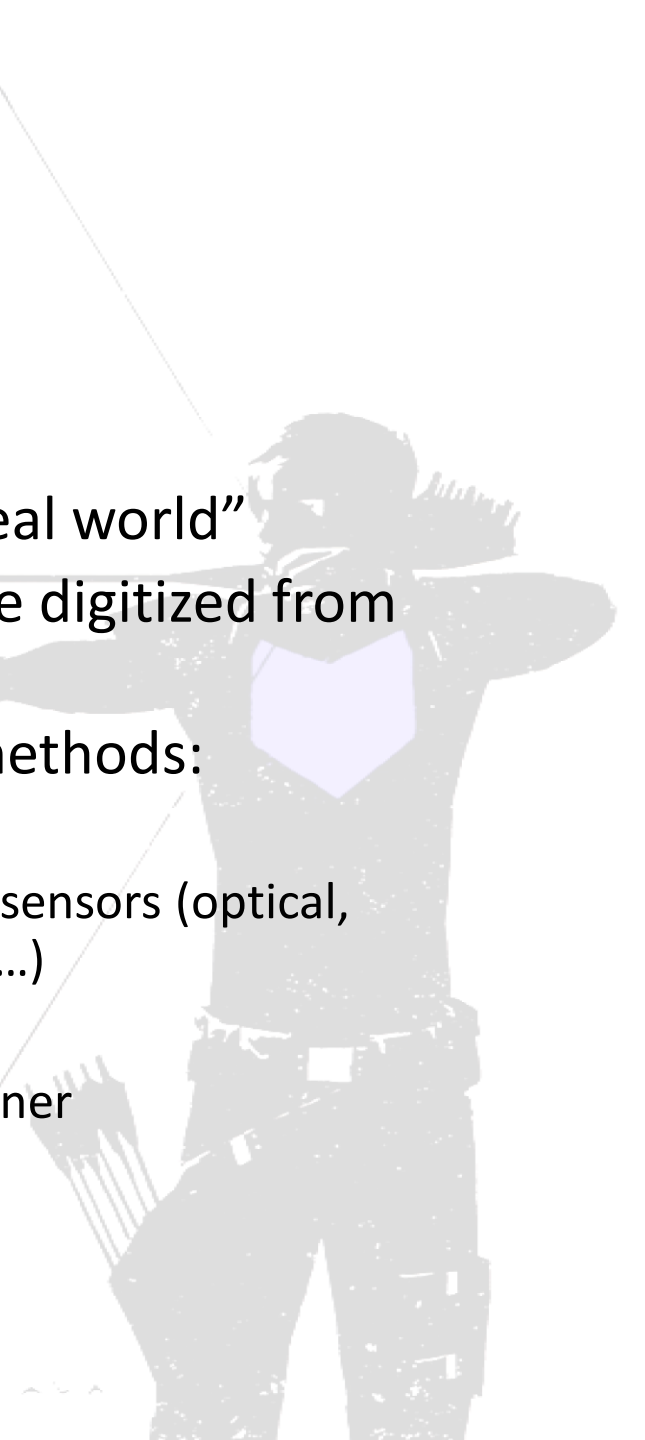
- The stages are
  - Image acquisisition – how we obtain images in the first place
  - Preprocessing – any effects applied before mapping (e.g., crop, mask, filter)
  - Mapping – catch-all stage involving image transformations or image composition
  - Postprocessing – any effects applied after mapping (e.g., texturizing, color remapping)
  - Output – printing or displaying on a screen
- In practice, stages may be skipped
- Middle stages are often interlaced



Stage 1: Image Acquisition

- Image Synthesis
  - Images created by a computer
  - Painted in 2D
    - Corel Painter
    - Photoshop
  - Rendered from 3D geometry
    - Pixar's RenderMan
    - Autodesk's Maya
    - Your projects
  - Procedurally textured
    - Generated images intended to mimic their natural counterparts
    - e.g., procedural wood grain, marble

- Image Capture
  - Images from the "real world"
  - Information must be digitized from an analog signal
  - Common capture methods:
    - Digital camera
    - Satellite data from sensors (optical, thermal, radiation,…)
    - Drum scanner
    - Flatbed photo scanner
    - Frames from video



Stage 2: Preprocessing

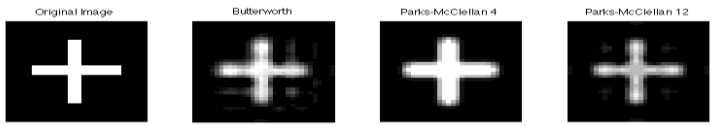
- Each source image is adjusted to fit a given tone, size, shape, etc., to match a desired quality or to match other images
- Can make a set of dissimilar images appear similar (if they are to be composited later), or make similar parts of an image appear dissimilar (such as contrast enhancement)

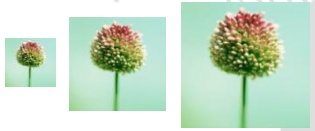
Original

Adjusted grayscale curve

Stage 2: Preprocessing

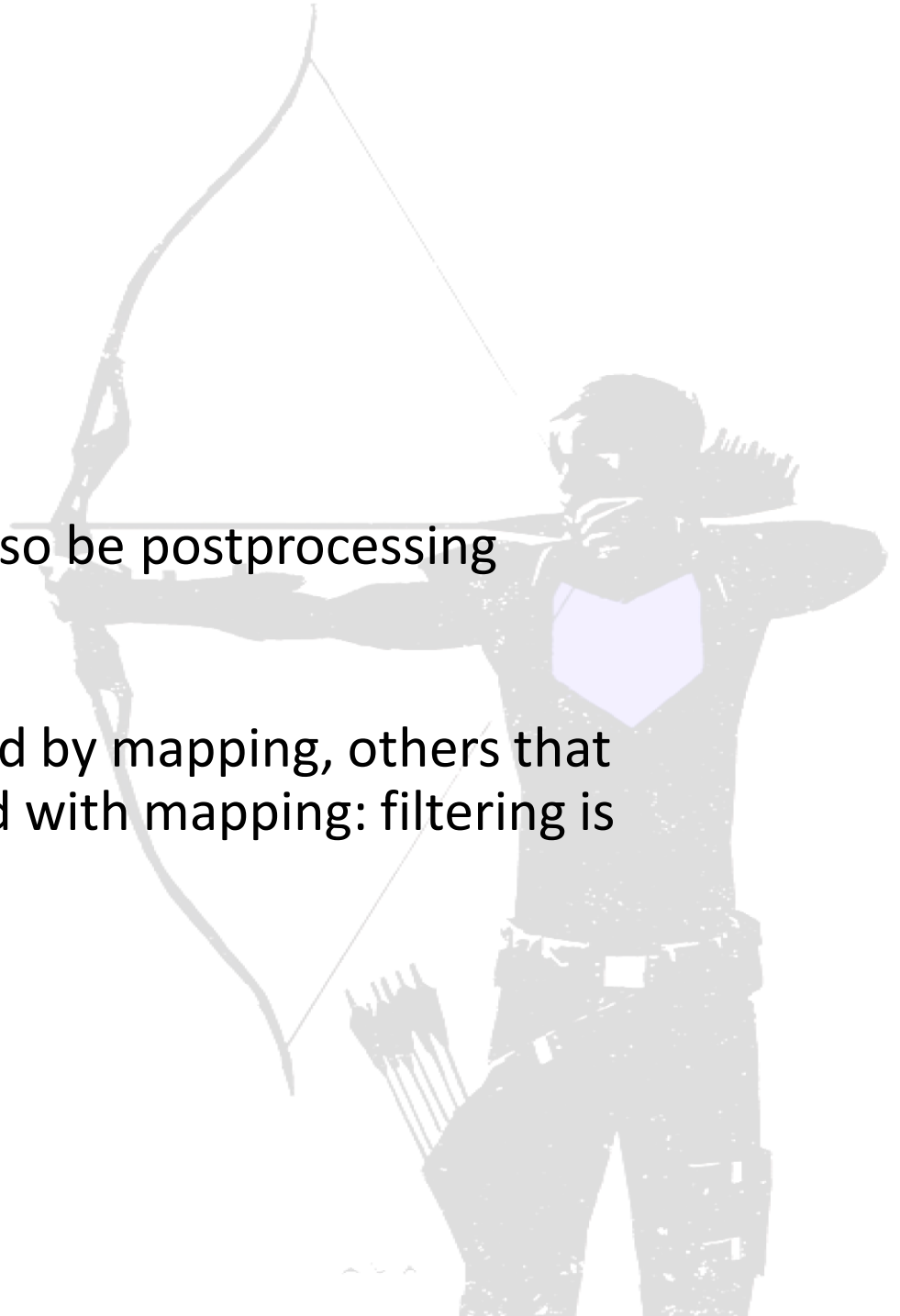
- Preprocessing techniques include:
  - Adjusting color or grayscale curve
  - Cropping
  - Masking (cutting out part of an image)
  - Blurring and sharpening
  - Edge detection/enhancement
  - Filtering and antialiasing
  - Scaling up (super sampling) or scaling down (sub sampling)





Stage 2: Preprocessing

- Notes:
  - Blurring, sharpening, and edge detection can also be postprocessing techniques
  - Some preprocessing algorithms are not followed by mapping, others that involve resampling the image may be interlaced with mapping: filtering is done this way



Stage 3: Mapping

- Mapping is a catch-all stage where several images are combined, or geometric transformations are applied
- Transformations include:
  - Rotating
  - Scaling
  - Shearing
  - Warping
  - Feature-based morphing
- Compositing:
  - Basic image overlay
  - Smooth blending with alpha channels
  - Poisson image blending
    - Seamlessly transfers “details” (like edges) from part of one image to another

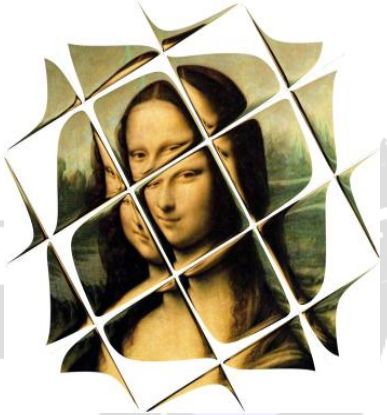


Image Warping

Poisson Image Blending

Image credit: © Evan Wallace 2010

Stage 4: Postprocessing

- Creates global effects across an entire image or selected area
- Art effects
  - Posterizing
  - Faked “aging” of an image
  - Faked “out-of-focus”
  - “Impressionist” pixel remapping
  - Texturizing
- Technical effects
  - Color remapping for contrast enhancement
  - Color to B&W conversion
  - Color separation for printing (RGB to CMYK)
  - Scan retouching and color/contrast balancing

Posterizing

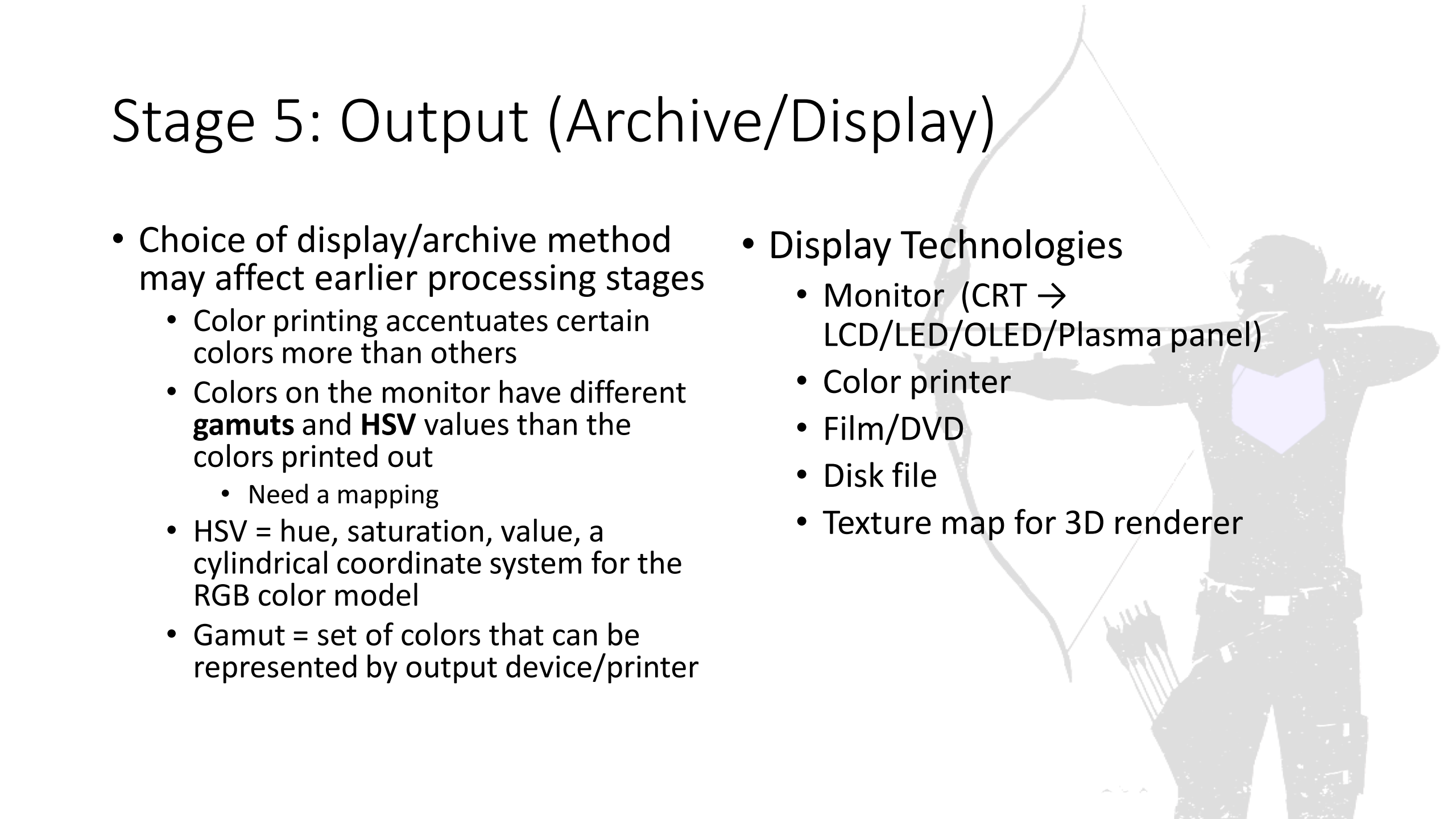
Aging

Impressionist

Stage 5: Output (Archive/Display)

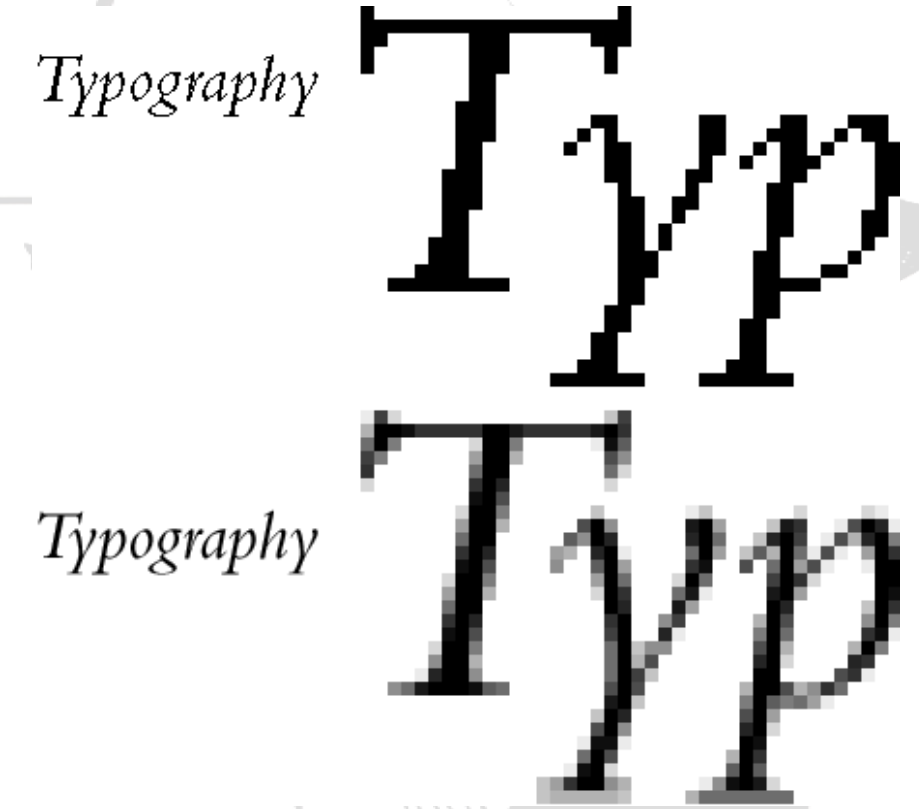
- Choice of display/archive method may affect earlier processing stages
  - Color printing accentuates certain colors more than others
  - Colors on the monitor have different **gamuts** and **HSV** values than the colors printed out
    - Need a mapping
  - HSV = hue, saturation, value, a cylindrical coordinate system for the RGB color model
  - Gamut = set of colors that can be represented by output device/printer

- Display Technologies
  - Monitor (CRT → LCD/LED/OLED/Plasma panel)
  - Color printer
  - Film/DVD
  - Disk file
  - Texture map for 3D renderer



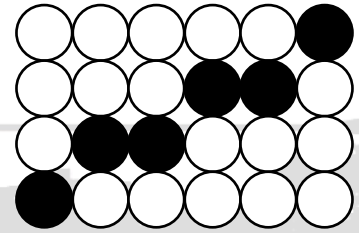
“Jaggies” & Aliasing

- “Jaggies” an informal name for artifacts from poorly representing continuous geometry by a discrete 2D grid of pixels – a form of aliasing (later…)
  - Jaggies are a manifestation of **sampling error** and loss of information (aliasing of high frequency components by low frequency ones)
- Effect of jaggies can be reduced by anti-aliasing, which smoothes out the pixels around the jaggies by averaging
  - Shades of gray instead of sharp black/white transitions
  - Diminishes HVS’ response to sharp transitions (Mach banding)

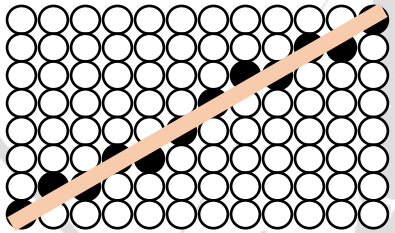


Representing lines:Point sampling, single pixel

- Midpoint algorithm: in each column, pick the pixel with the closest center to the line
  - A form of point sampling: sample the line at each of the integer X values
  - Pick a single pixel to represent the line's intensity, full on or full off
- Doubling resolution in x and y only lessens the problem, but costs 4 times memory, bandwidth, and scan conversion time!
- Note: This works for -1 < slope < 1, use rows instead of columns for the other case or there will be gaps in the line

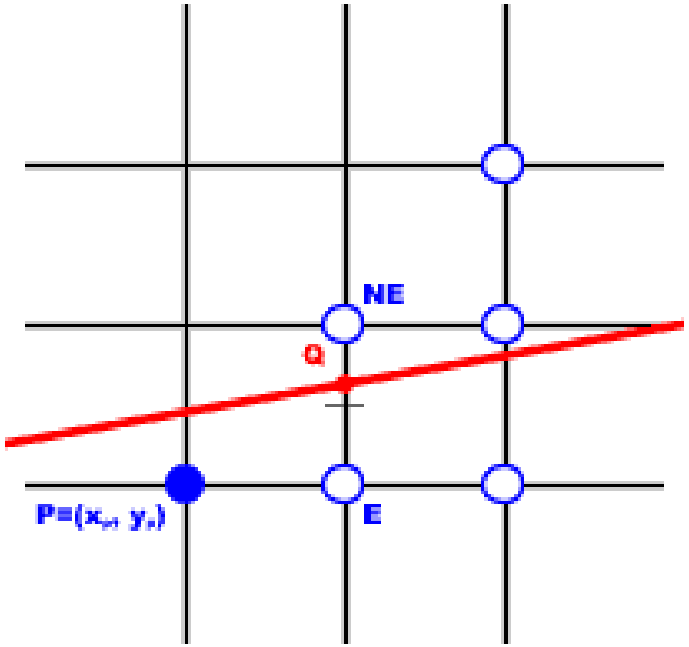


Line approximation using point sampling

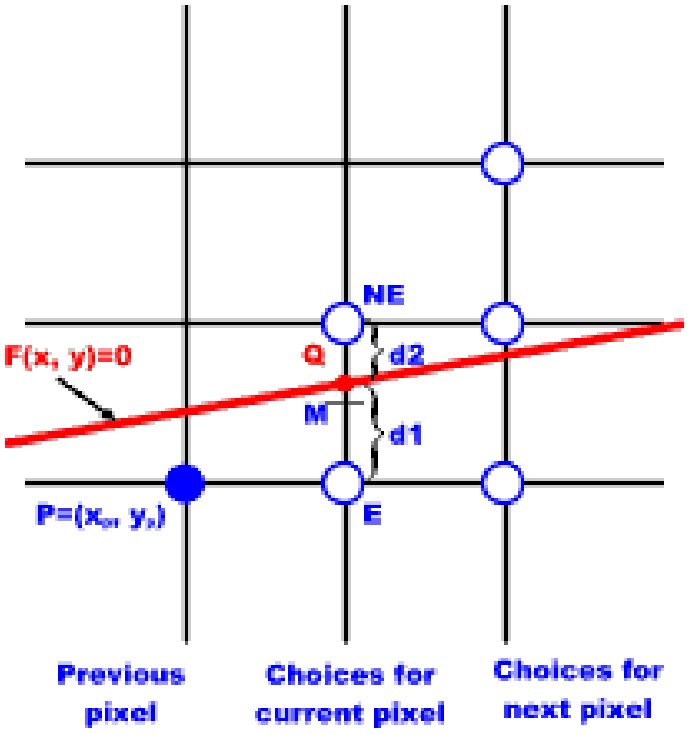


Approximating same line at 2x the resolution

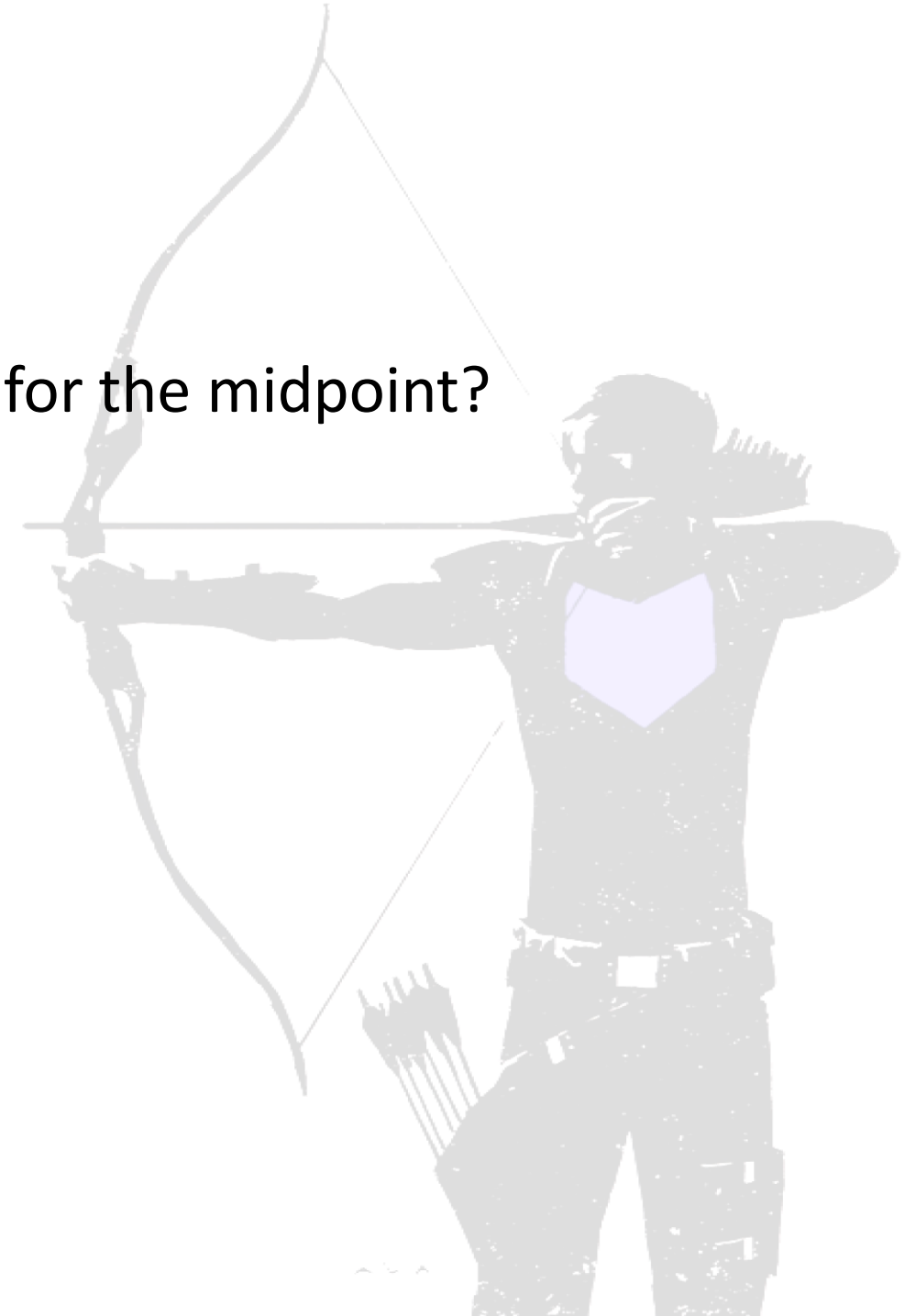
Midpoint Line Algorithm



Midpoint Line Algorithm



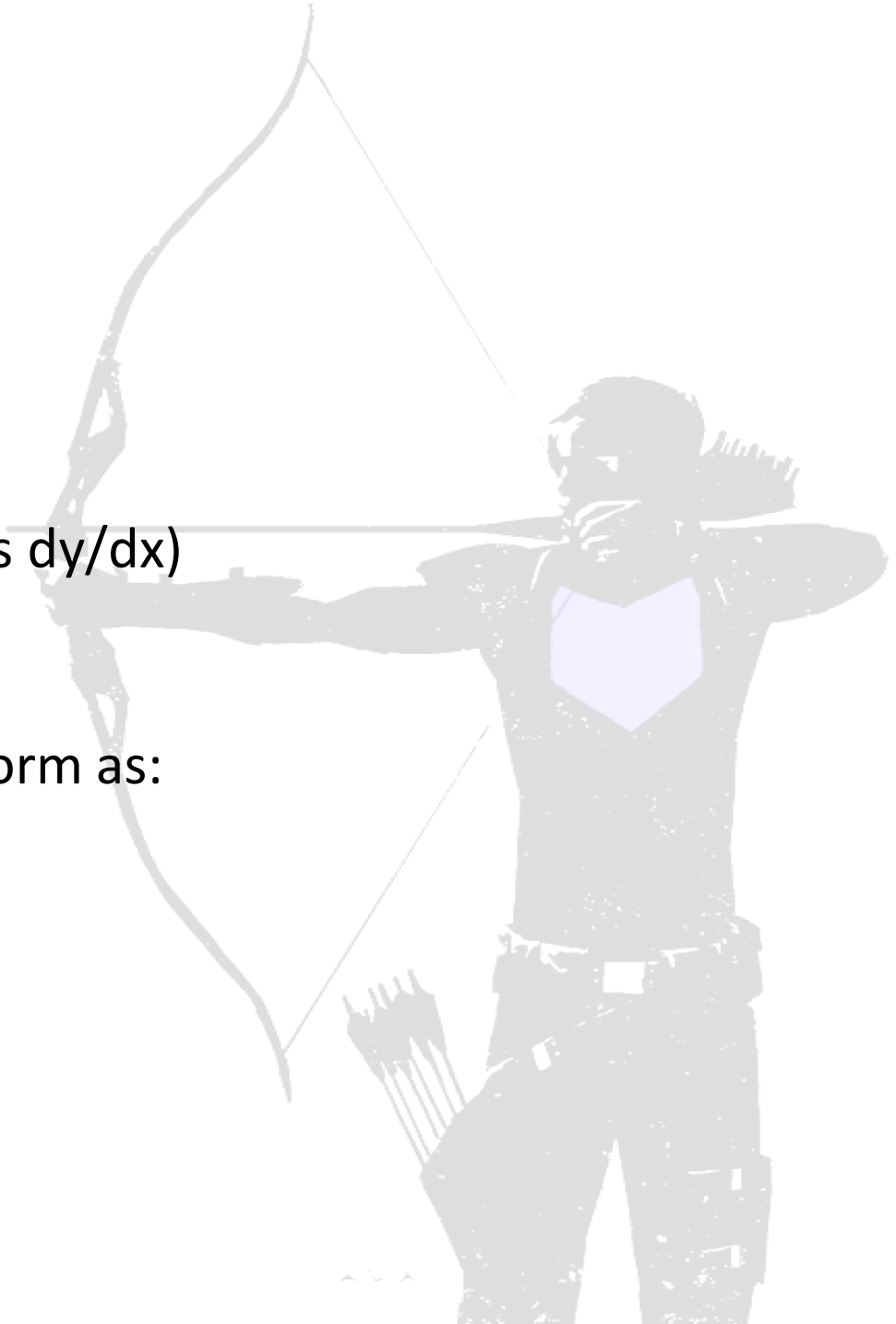
- Is F(M) < 0 for the midpoint?
- Is d1 < d2?



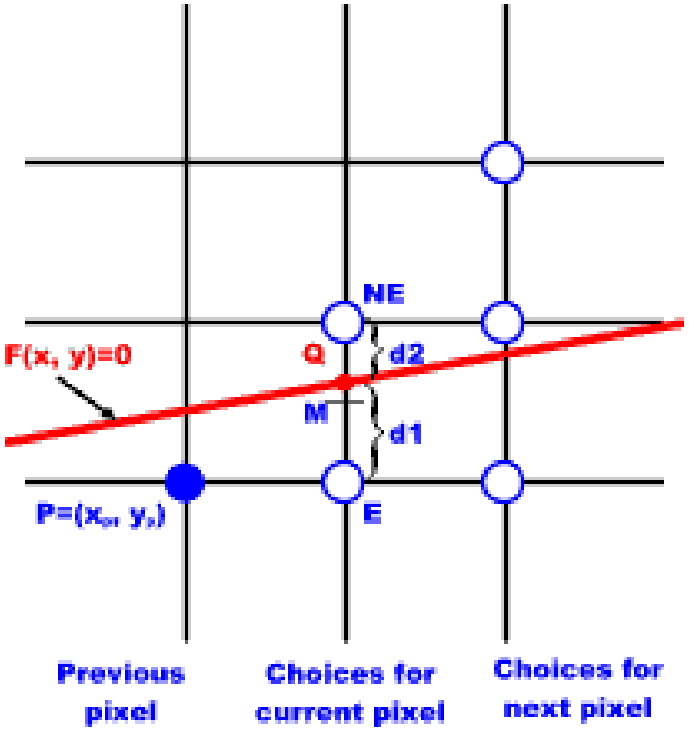
Midpoint Line Algorithm

- Equations:
  - Slope intercept form – $y = mx + b$ (where m is dy/dx)
  - Standard form of a line – $F(x,y) = ax + by + c = 0$
  - Combining these we can rewrite the standard form as:
    $F(x,y) = dy\ x - dx\ y + B\ dx = 0$

    where $a = dy$, $b = dx$, and $c = B\ dx$

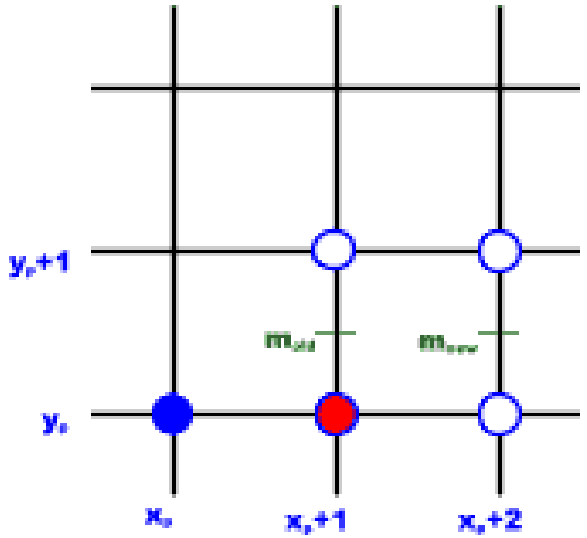


Midpoint Line Algorithm



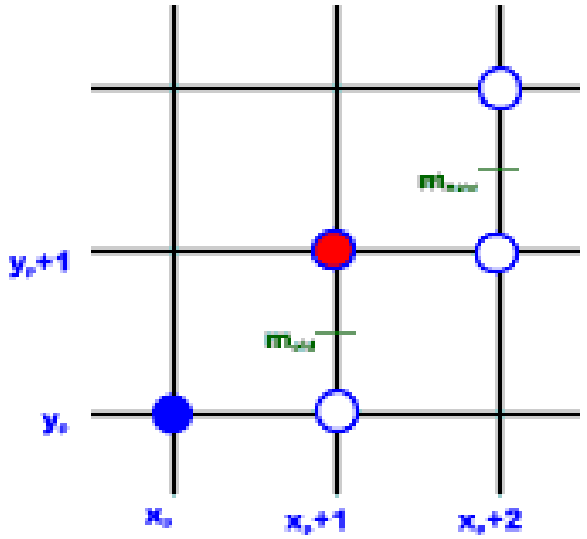
- Determining our distance for the midpoint:
  - $d = F(m) = F(x_p + 1, y_p + ½)$
- if $d > 0$
  choose NE
  else
  choose E

Midpoint Line Algorithm



- If we choose E
  - Increment our midpoint by 1 in x
  - $d_{new} = F(m_{new}) = F(x_p + 2, y_p + ½)$
  - deltaE = $d_{new} – d_{old} = a = dy$

Midpoint Line Algorithm

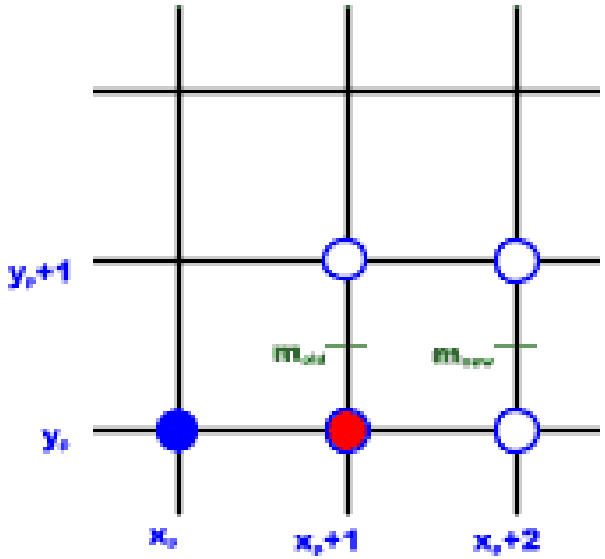


- If we choose NE
  - Increment our midpoint by 1 in x and y
  - $d_{new} = F(m_{new}) = F(x_p + 2, y_p + 1½)$
  - deltaNE = $d_{new} – d_{old} = a + b = dy - dx$

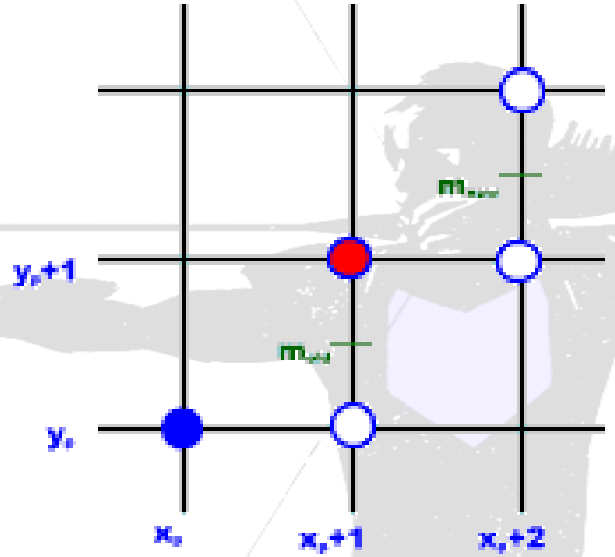
Midpoint Line Algorithm

- Calculating our first midpoint:
  - $d_{start} = F(x_0 + 1, y_0 + ½)$
- As a standard line, this looks like:
  - $a\ x_0 + a + b\ y_0 + ½\ b + c$
    $= F(x_0, y_0) + a + ½\ b$
    $= d_{new} – d_{old}$
    $= a + ½\ b$
    $= dy – ½\ dx$

Midpoint Line Algorithm



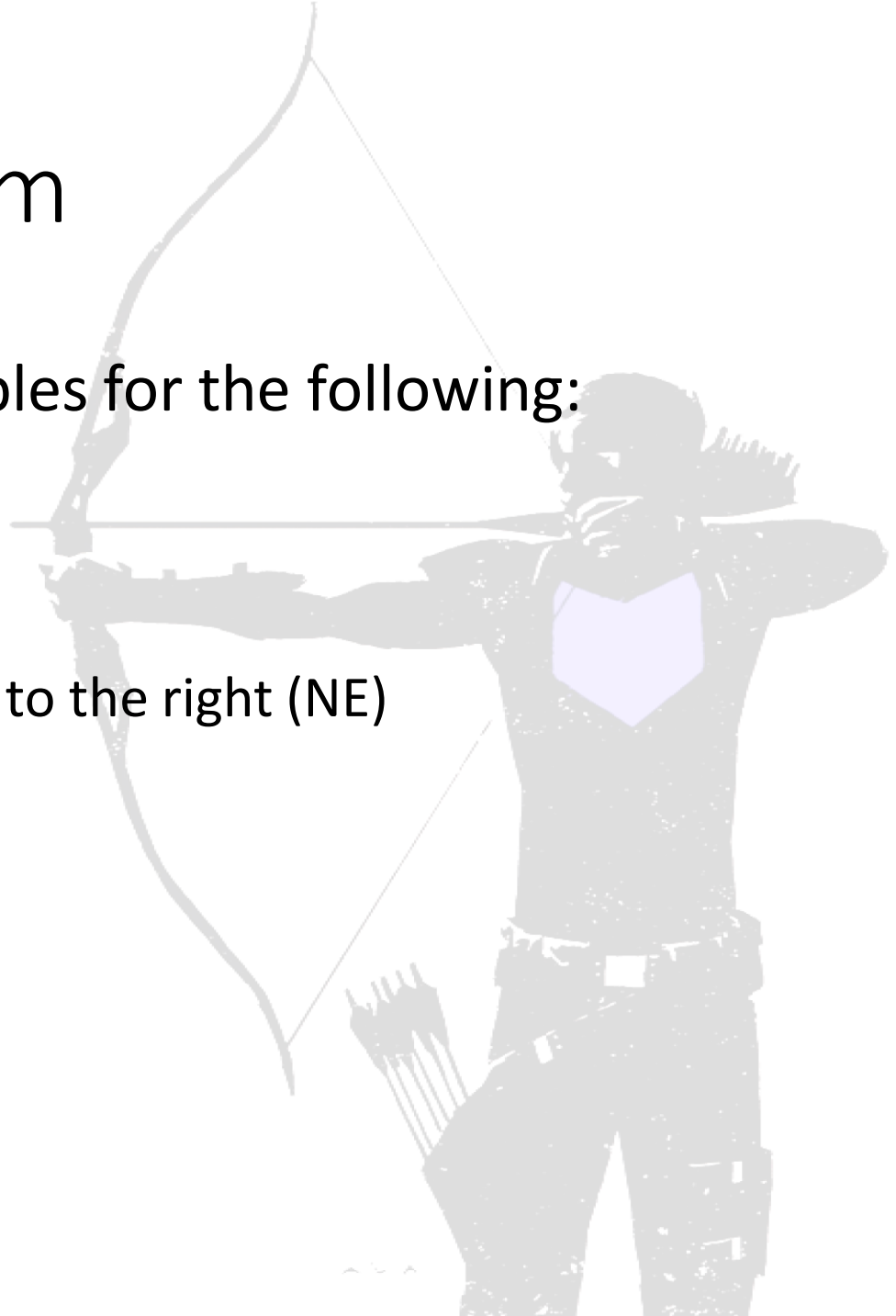
- Calculating our first midpoint
  - $d_{start} = F(x_0 + 1, y_0 + ½)$
  - $= 2(dy – ½\ dx)$
  - $= 2\ dy – dx$



- Case E
  - Increment our midpoint by 1 in x
  - $d_{new} = F(m_{new}) = F(x_p + 2, y_p + ½)$
  - $deltaE = d_{new} – d_{old} = a = 2\ dy$
- Case NE
  - Increment our midpoint by 1 in x and y
  - $d_{new} = F(m_{new}) = F(x_p + 2, y_p + 1½)$
  - $deltaNE = d_{new} – d_{old} = a = 2\ (dy – dx)$

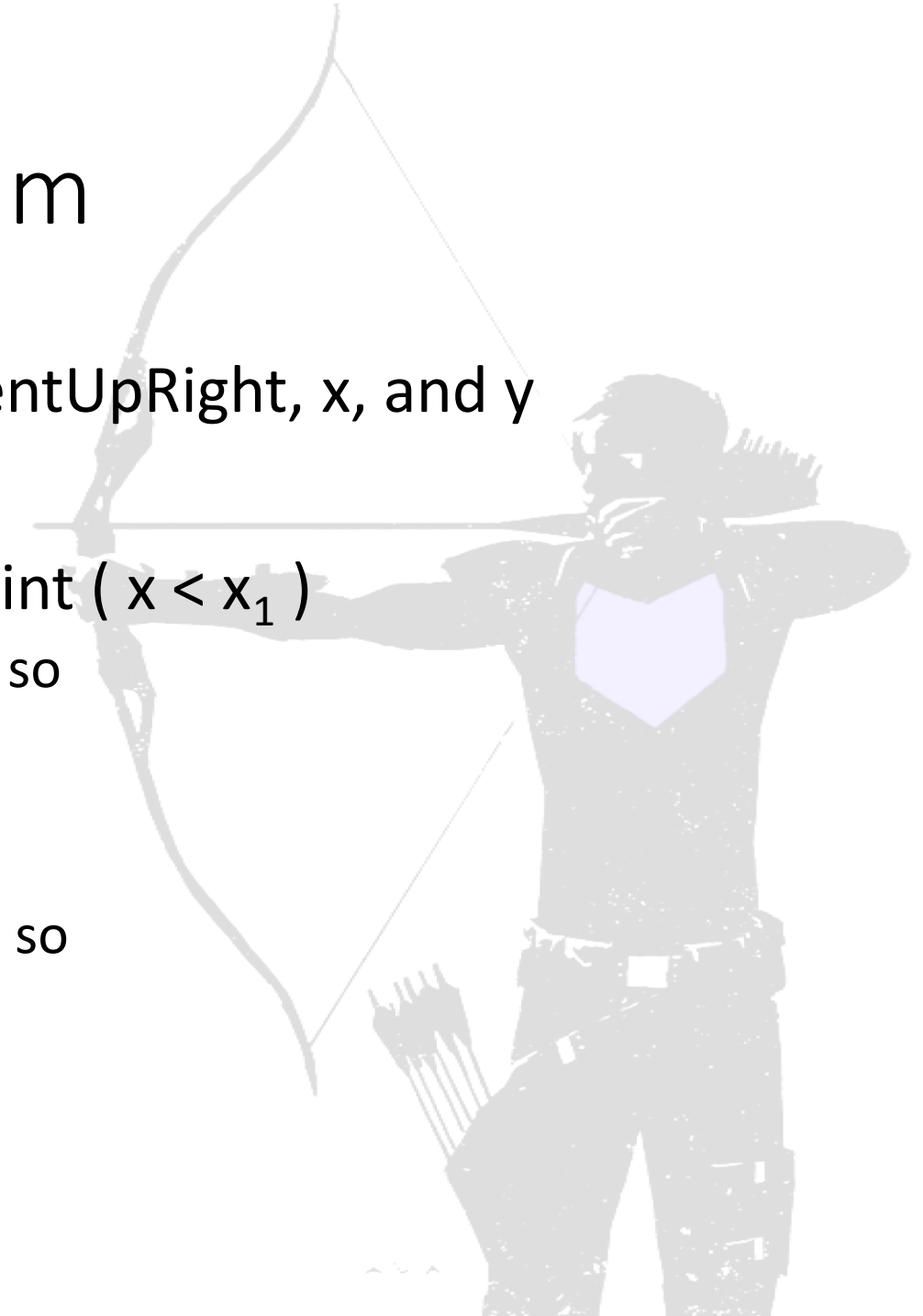
Psuedocode for Line Algorithm

- Given the endpoints of a line, declare variables for the following:
  - dx - change in x
  - dy - change in y
  - incrementRight - how much to move right (E)
  - incrementUpRight - how much to move up and to the right (NE)
  - d - the distance to the midpoint
  - x - the current x value to plot
  - y - the current y value to plot



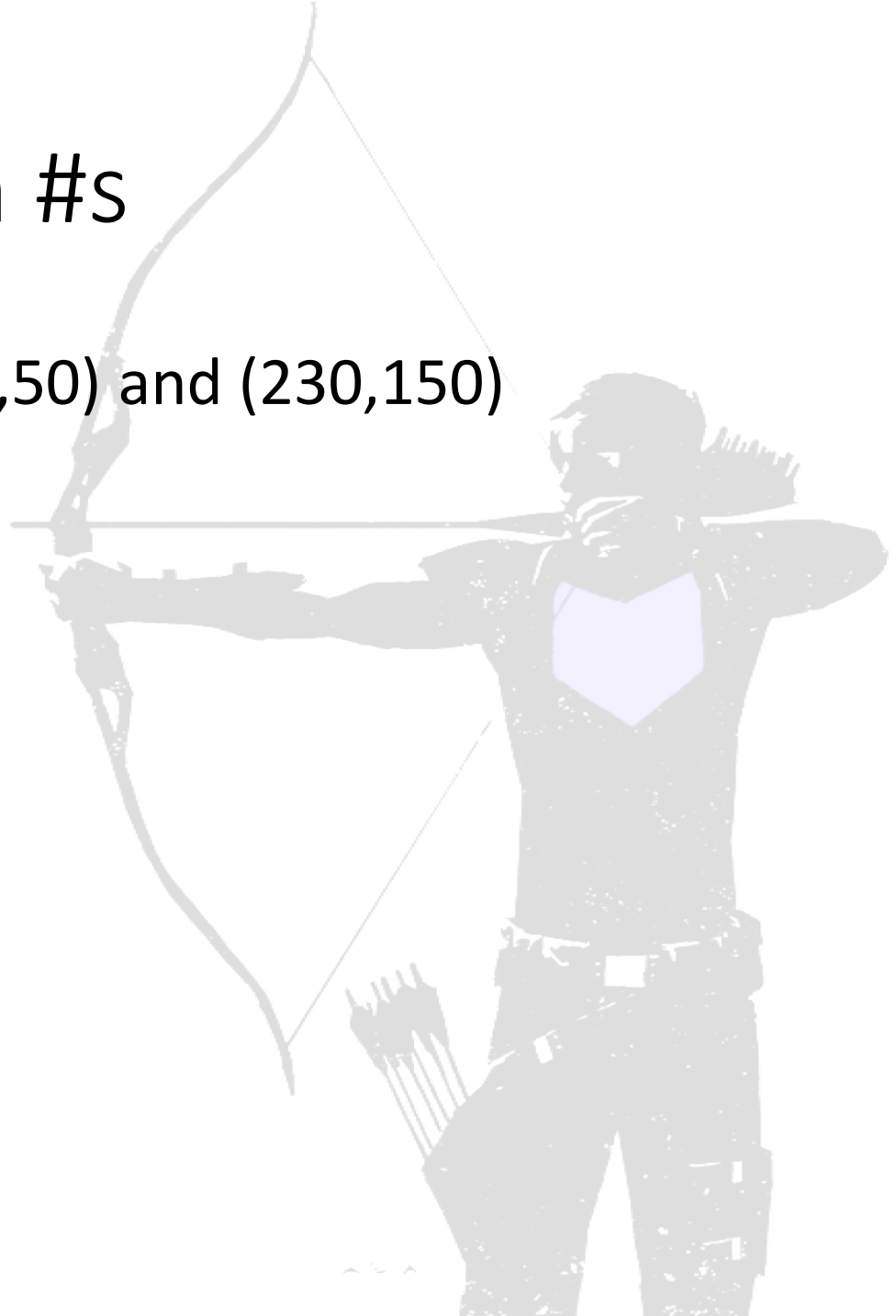
Psuedocode for Line Algorithm

- Initialize dx, dy, d, incrementRight, incrementUpRight, x, and y
- Draw the first endpoint
- While we haven't drawn the second endpoint ($x < x_1$)
  - If d > 0 we choose to move up and to the right so
    d += incrementUpRight
    x increases by 1
    y increases by 1
  - Otherwise we choose to move just to the right so
    d += incrementRight
    x increases by 1
  - Draw the new point



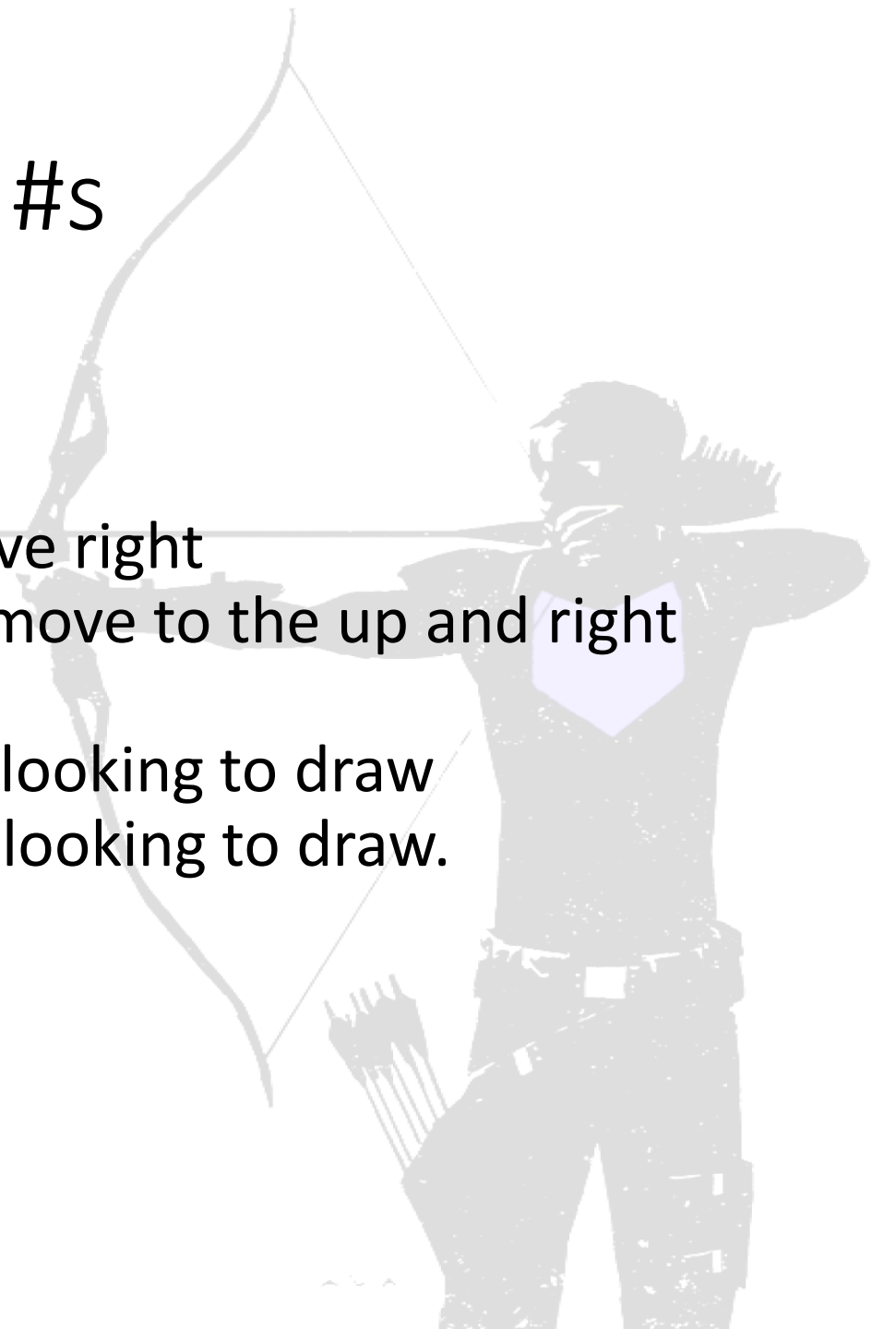
Midpoint Line Algorithm with #s

- We are given the following end points – (50,50) and (230,150)
- We can picture our points like this
  $x_0 = 50$, $x_1 = 230$, $y_0 = 50$, and $y_1 = 150$
- We need to initialize the following:
  dx,
  dy,
  incrementRight,
  incrementUpRight,
  d,
  x,
  and y.



Midpoint Line Algorithm with #s

- dx is our change in x
  dy is our change in y
  incrementRight is how much we need to move right
  incrementUpRight is how much we need to move to the up and right
  d is the distance to our midpoint
  x is the x value to the point we are currently looking to draw
  y is the y value to the point we are currently looking to draw.

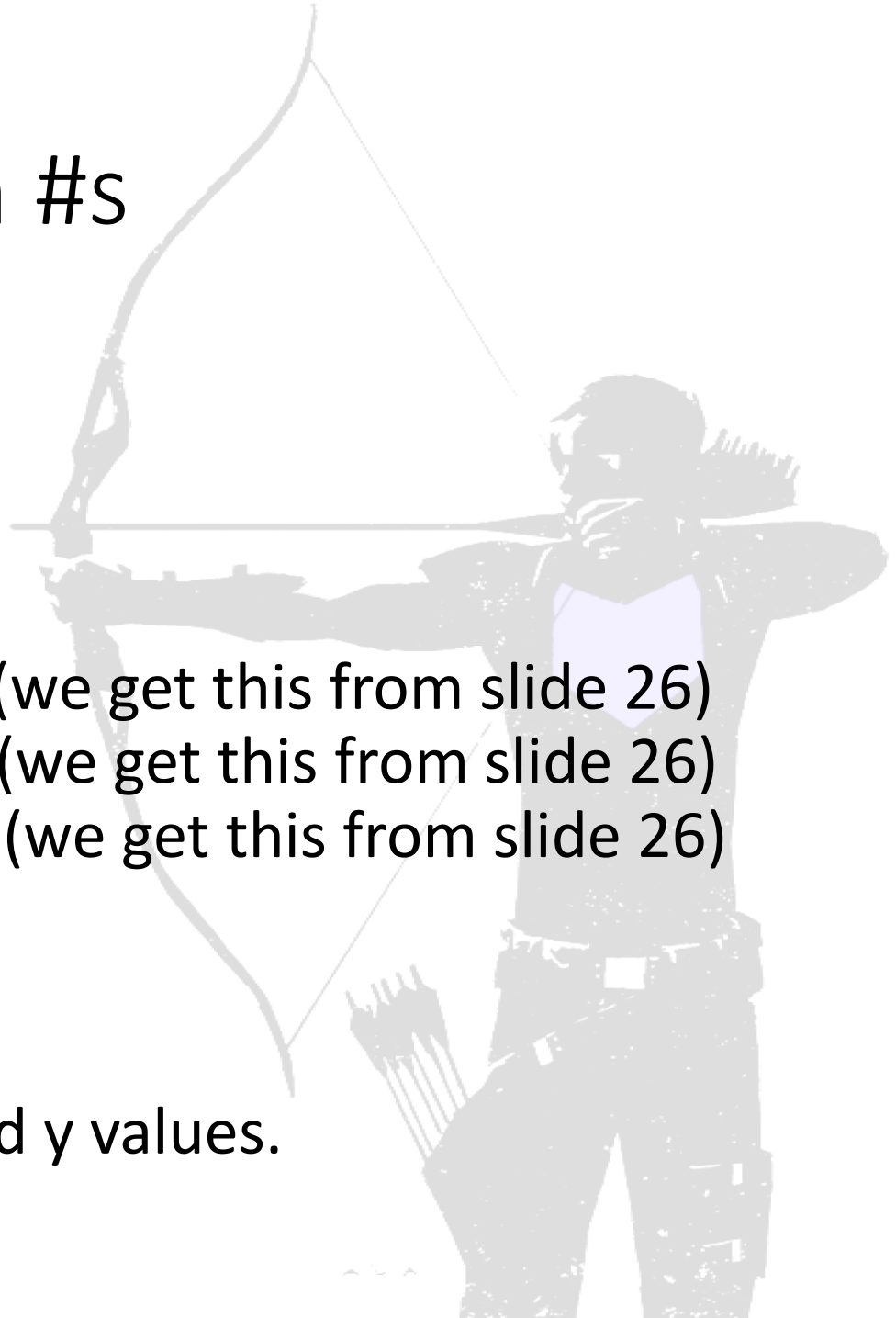


Midpoint Line Algorithm with #s

- Therefore:

  dx = 230 - 50 = 180
  dy = 150 - 50 = 100
  d = 2*(100) - 180 = 20 (we get this from slide 26)
  incrementRight = 2*(100) = 200 (we get this from slide 26)
  incrementUpRight = 2*(100-180) = -160 (we get this from slide 26)
  x = 50
  y = 50

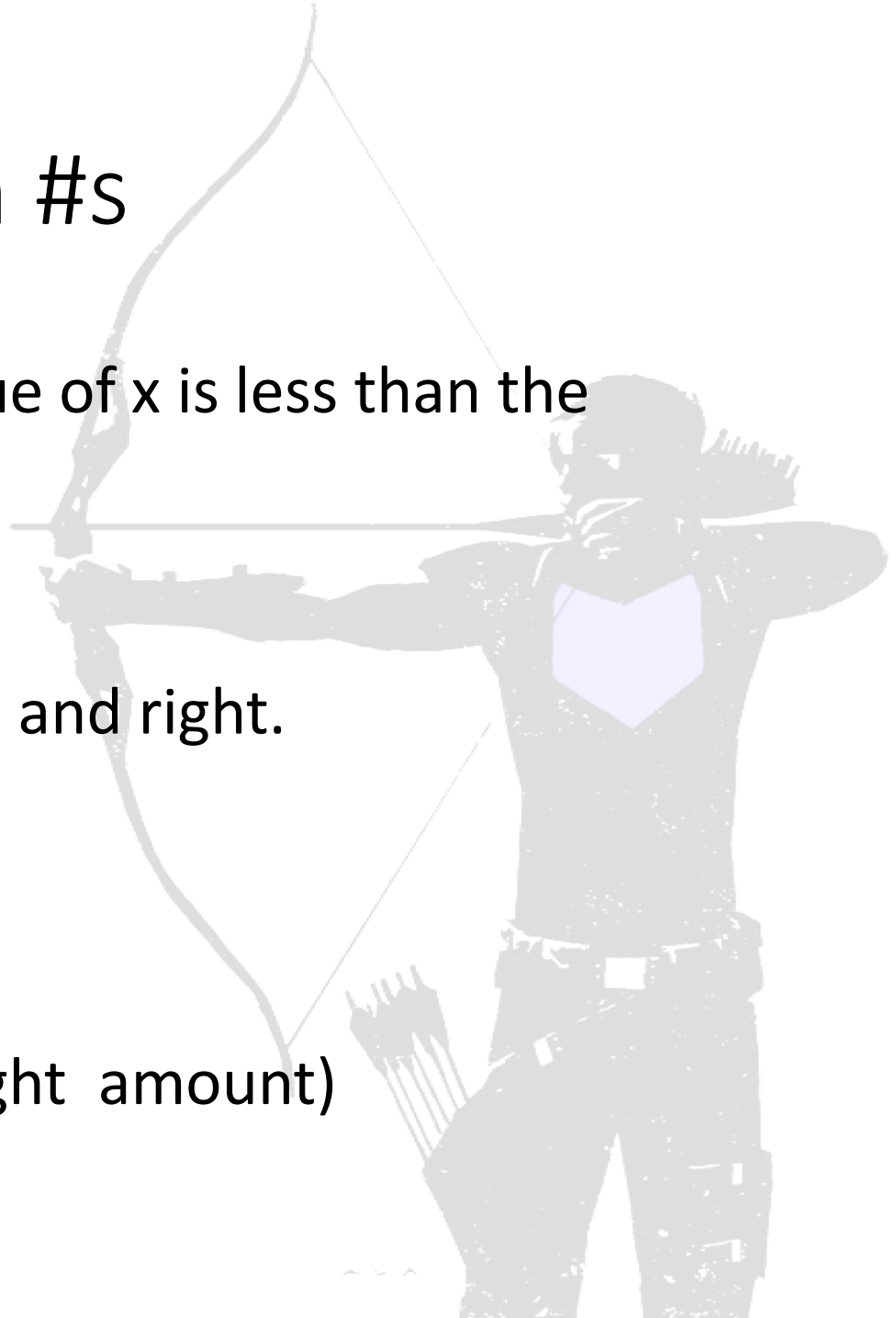
  Now we can draw our first point at our x and y values.



Midpoint Line Algorithm with #s

- Our loop now begins, while our current value of x is less than the endpoint x_1 we repeat the following:

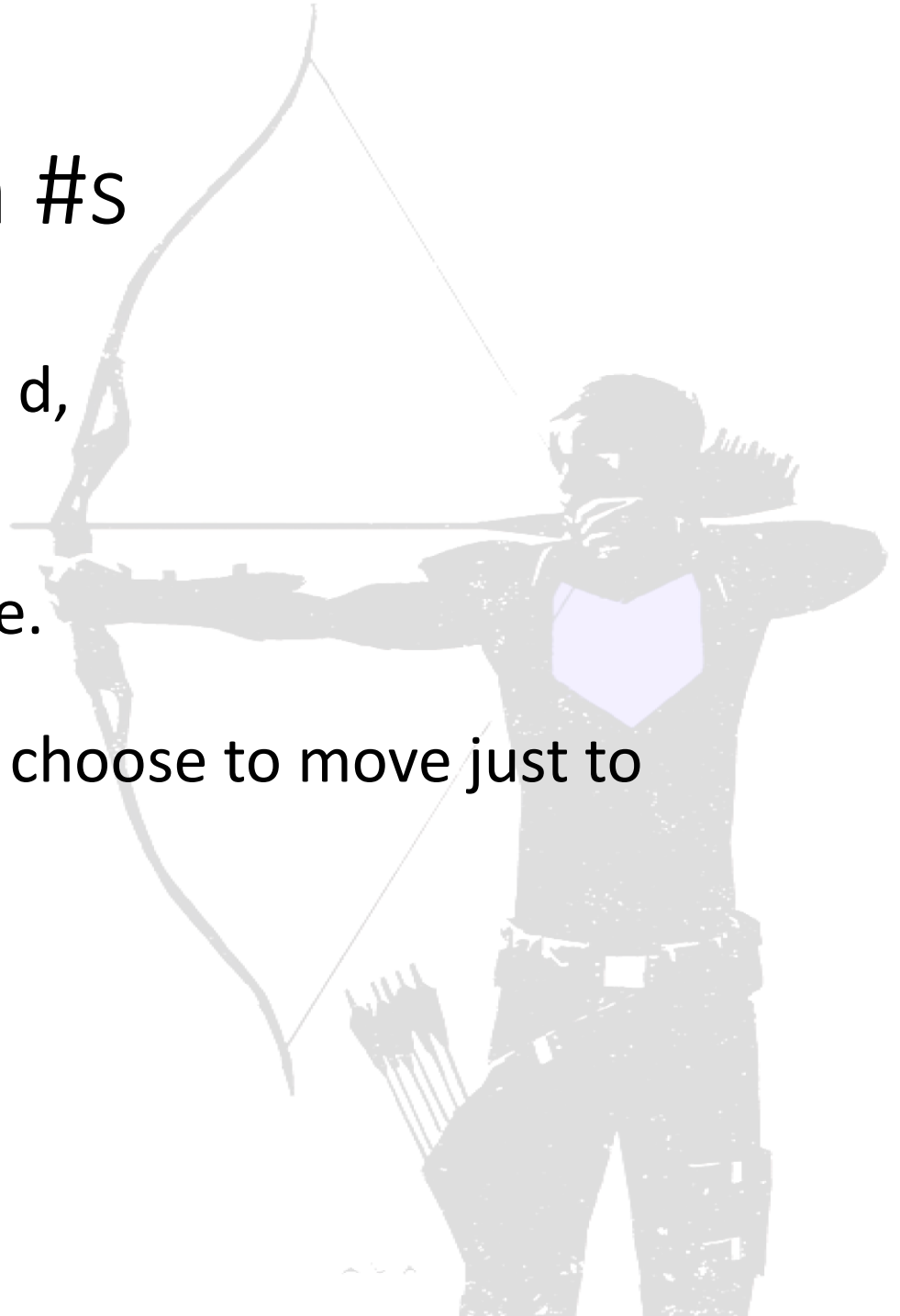
- We look at our distance, d.
  d is 20, which is positive so we will move up and right.
  x = 50 + 1 = 51 (moved right)
  y = 50 + 1 = 51 (moved up)
  Let's draw our new point at (51, 51)
  Our d will now be: d = 20+(-160) = -140
  (since we increase it by our incrementUpRight amount)



Midpoint Line Algorithm with #s

- Now we go back to loop actions of checking d,
  incrementing our x and y values as needed,
  drawing our new point,
  and incrementing d by the appropriate value.

  Our d is now -140 which is negative so we'll choose to move just to the right.



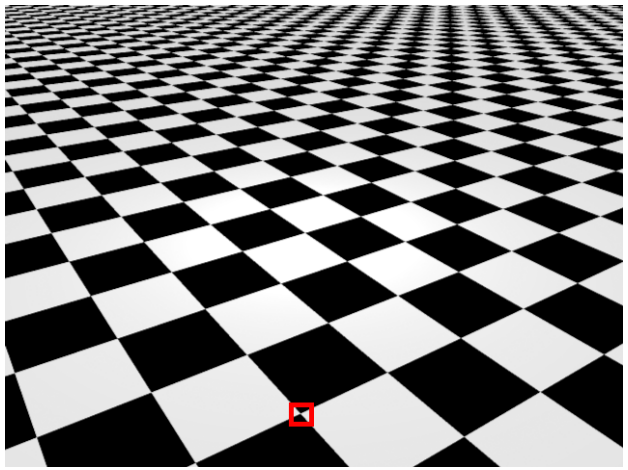
Midpoint Line Algorithm with #s

- x = 51 + 1 = 52 (moved right)
  y = 51 + 0 = 51 (did not move)
  we draw the next point at (52, 51)

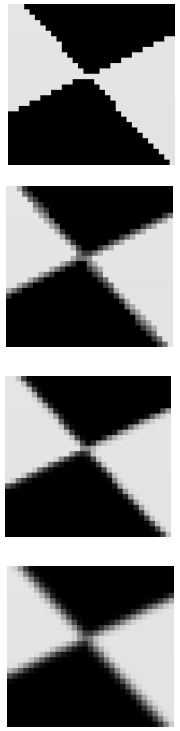
  Our d will now be: d = -140 + 200 = 60

- Repeat until we've passed out endpoint

Anti-Aliasing Example



Checkerboard with Supersampling



Close-up of original, aliased render

Antialiasing Techniques:

Blur filter – weighted average of neighboring pixels

Supersampling - sample multiple points within a given pixel and average the result

Supersampling and Blurring

References

- Computer Graphics: Principles and Practice 3rd Ed. (Our Textbook)
  - Parts of Chapters 7 & 18
- Slides from Andries van Dam of Brown University (one of the authors of our Textbook)
- http://www.csee.umbc.edu/~rheingan/435/pages/res/gen-2.Lines-single-page-0.html